Hacking y Pentesting con Python

Módulo 4: Networking con Python.

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Módulo 4: Networking con Python.

PRESENTACIÓN Y OBJETIVOS

El objetivo de este módulo es el de enseñar las posibilidades que ofrece Pyhon y Scapy a la hora de analizar entornos de red y crear paquetes de datos que se puedan enviar a un destino concreto. Se podrá ver cómo utilizar la "socket API" disponible en el lenguaje y el uso de la librería Scapy, la cual se ha convertido en una dependencia de múltiples proyectos orientados a la ciberseguridad y el hacking ético.

Objetivos

- Conocer los elementos disponibles en el lenguaje para la creación de sockets y establecimiento de conexiones con otros sistemas.
- Aprender a utilizar Scapy para el reconocimiento activo de redes y su posterior análisis.
- Enseñar los fundamentos para la creación de herramientas para el análisis de redes con Scapy.

Sockets en Python.

Un socket es un punto que permite la conexión entre procesos que se pueden encontrar en cualquier ubicación en la red. Está compuesto por una dirección y un número de puerto, además son únicos, es decir, si un socket se encuentra referencia con una IP/dominio y un puerto, no puede crear otro socket utilizando esta misma información.

Por otro lado, existen sockets que actúan como servidores o como clientes y dependiendo de esto, las primitivas que se deben ejecutar en el sistema operativo son diferentes.

Finalmente, los sockets siguen especificaciones concretas que todos los sistemas operativos deben seguir con el objetivo de fomentar la interoperabilidad entre los puntos de conexión y evitar fallos.

Lenguajes de programación como Python se basan en las funciones definidas en el sistema operativo y dado que todos siguen la misma nomenclatura de nombres, hace que sea fácil la creación de Sockets en cualquier sistema que tenga Python instalado.

1. Server socket

En el caso de un socket del tipo “server”, el punto se encontrará en estado de espera para recibir una conexión por parte de un cliente. En cuando dicho conexión se produce, el socket “server” tendrá información del socket “client” para poder intercambiar paquetes de datos. El procedimiento para crear un socket “server” es el siguiente:

- Creación del objeto socket correspondiente. Hay que tener en cuenta que puede ser del tipo TCP o UDP.
- Invocar a la función “bind” que se encargará de verificar si el puerto indicado se encuentra disponible y si ese es el caso, reservarlo para el socket que se ha creado en el sistema operativo.
- Invocar a la función “listen” la cual permite especificar el número máximo de conexiones concurrentes.
- Invocar a la función “accept” la cual se encarga de aceptar una conexión entrante por parte de un cliente.
- Invocar a la función “send” la cual permite enviar un paquete de datos a otro extremo de la conexión.
- Invocar a la función “recv” la cual permite recibir un paquete de datos con un tamaño fijo desde el otro extremo de la conexión.
- Invocar a la función “close”, la cual se encarga de notificar al otro extremo que la conexión se va a cerrar.

Como se puede comprobar, el flujo no es muy complejo y como se verá a continuación, hay funciones comunes también para el lado del cliente, sin embargo la creación de un socket cliente es mucho más simple.

2. CLIENT SOCKET.

En el caso de un socket cliente, es normalmente el sistema operativo el que se encarga de seleccionar de forma aleatoria el puerto del socket, sin embargo es posible cambiar este comportamiento de forma programática desde Python si es necesario, en cualquier caso las invocaciones que se llevan a cabo en un socket cliente son mucho más simples, ya que el cliente lo único que hace es establecer una conexión con un servidor e intercambiar paquetes de datos.

El procedimiento para crear un socket "client" es el siguiente:

- Creación del objeto socket correspondiente. Hay que tener en cuenta que puede ser del tipo TCP o UDP.
- Invocar a la función "connect" especificando la IP o dominio del servidor y su puerto. Esta instrucción se encargará de realizar la conexión con dicho socket servidor y si se encuentra disponible se llevan a cabo los siguientes pasos, en caso contrario se lanzará una excepción indicando que no se ha podido establecer la conexión.
- Invocar a la función "send" la cual permite enviar un paquete de datos a otro extremo de la conexión.
- Invocar a la función "recv" la cual permite recibir un paquete de datos con un tamaño fijo desde el otro extremo de la conexión.
- Invocar a la función "close", la cual se encarga de notificar al otro extremo que la conexión se va a cerrar.

En el caso de Python, el módulo "socket" es el indicado para la creación de sockets. Incluye todas las funciones necesarias para crear sockets del tipo TCP/UDP en modo cliente y servidor. El script 4-sockets.py demuestra el uso de dicho módulo y permite ver cómo crear un servidor y un cliente TCP desde el mismo programa.

Hay que tener en cuenta que en el punto de anclaje del programa se encuentran definidas invocaciones a las funciones “Server” y “Client”. Para probar su funcionamiento, se debe ejecutar primero el script con la función “Server” y posteriormente, comentar dicha función y ejecutarlo con la función “Client” habilitada.

Recuerda

Los sockets NO son una característica propia de Python. Se trata de un concepto global que se implementa todos los sistemas operativos modernos de forma nativa. Lenguajes de programación de alto nivel como Python, simplemente ponen a disposición de los desarrolladores una API para acceder a las funciones de red habilitadas en el sistema operativo.

Reconocimiento y enumeración con Scapy.

Scapy es una librería en Python que se caracteriza por ser muy potente y flexible a la hora de capturar, analizar, manipular, e inyectar paquetes en un segmento de red. Soporta una gran cantidad de protocolos de red y se utiliza en múltiples proyectos de seguridad informática en los que se requiere manipular paquetes de red.

Además de permitir la manipulación de paquetes, también es una excelente herramienta para realizar actividades de reconocimiento y escaneo de puertos, con la ventaja evidente de que permite controlar todo desde un script en Python.

Por otro lado, independientemente de la herramienta utilizada, a lo largo de los años se han ido desarrollando técnicas que permiten realizar escaneos de puertos y detectar servicios en ejecución, puertos cerrados y filtrados.

Cada una tiene características muy concretas y para un atacante o pentester, es importante conocerlas bien para saber en qué momento resulta conveniente utilizarlas.

3. USO DE SCAPY.

Con Scapy no es necesario aprender un lenguaje de programación nuevo, solamente es necesario tener conocimientos sobre la sintaxis y estructura de los programas escritos en Python.

Su uso es simple, puede hacerse de forma interactiva (del mismo modo que se hace con el intérprete de Python) o directamente desde una rutina completa de código utilizando un fichero.

El funcionamiento básico de Scapy es simple, se crean uno o varios paquetes, se envían a un destino se capturan de respuestas emitidas.

Permite un alto grado de control en los paquetes enviados y en las respuestas recibidas. El trabajo analítico e interpretativo queda a criterio del programador, por lo tanto dicha persona debe de tener un buen nivel de conocimientos sobre protocolos de red y paquetes de datos.

La creación de paquetes con scapy sigue el modelo OSI, es decir, se deben de crear los paquetes partiendo de las capas más bajas hasta llegar a la capa de aplicación.

Por otro lado, se puede utilizar Scapy desde el intérprete diseñado para interactuar con su API o directamente desde cualquier script en Python, siendo la primera alternativa la más habitual a la hora de aprender el uso de las funciones que se encuentran disponibles en la librería y realizar pruebas.

Figura 3.1: Funciones disponibles en Scapy

Algunas de las funciones básicas en Scapy se listan a continuación:

ls() Lista los protocolos soportados por scapy.

lsc() Lista las funciones disponibles en scapy.

ipPacket = IP() Creación de un paquete IP con todos los valores por defecto.

packet = Ether()/IP(dst="google.com")/ICMP()/"ABCD" Creación de un paquete del tipo ICMP. Como se puede apreciar es necesario declarar cada una de las capas que componen le componen. En este caso, es ha sido necesaria una capa Ethernet, IP, ICMP y finalmente datos raw en la capa de aplicación.

ls(packet) La función ls también enseñar la estructura de un paquete determinado. Incluye la información de cada una de las capas del paquete.

sendp(packet) Función que permite enviar un paquete a su correspondiente destino sin esperar respuesta.

sendp(paquet, loop=1, inter=1) Con las opciones inter y loop de la función sendp es posible enviar el paquete de forma indefinida cada N segundos.

srp1(packet) Permite enviar y recibir paquetes, en este caso espera hasta obtener la primera respuesta por parte del destino.

packet.show()
packet.summary() Con estas funciones es posible ver la estructura del paquete recibido en un formato mucho más claro y simplificado.

pkts = sniff(iface="wlan0", count=3) Con la función sniff es posible capturar paquetes del mismo modo que lo hacen herramientas como tcpdump o wireshark. El resultado se almacena en una lista, en la cual se encontrarán cada uno de los paquetes capturados por la herramienta.

wrpcap("demo.pcap", pkts) Si se quiere almacenar un conjunto de paquetes en un fichero PCAP, se puede hacer usando la función wrpcap.

readed = rdpcap("demo.pcap") Con la función rdpcap se puede leer un fichero pcap y obtener un listado de paquetes, los cuales se almacenarán en un objeto lista y podrán ser manejados desde python.

Scapy soporta el formato BPF (Berkeley Packet Filters) el cual es un formato estándar para aplicar filtros sobre paquetes de red. Estos filtros pueden aplicarse sobre un conjunto de paquetes o directamente sobre una captura activa.

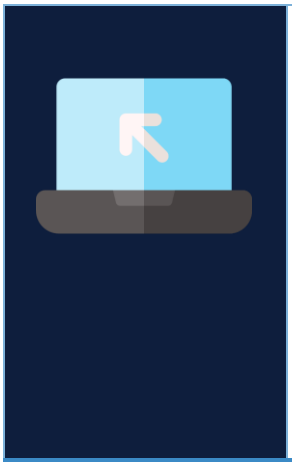
icmpPkts = sniff(iface="wlan0", filter="icmp" count=3) en esta instrucción, por ejemplo, se capturan exactamente 3 paquetes del tipo ICMP.

Scapy depende de la configuración de red que tenga definida, en ella se pueden especificar las rutas de acceso a otras redes, la interfaz de red por defecto que se usará para enviar los paquetes a un destino, entre otras cosas. Para acceder a dicha configuración se cuenta con el objeto **conf,** el cual se puede consultar o manipular desde el intérprete de scapy o desde cualquier script en Python.

conf.route Enseña las rutas definidas.

conf.route.add[net="192.168.2.0/24", gw="192.168.2.1"] Crea una nueva ruta con los valores indicados en “net” y “gw”.

conf.route.resync() Se encarga de indicar a scapy que debe sincronizar su configuración con el entorno en donde se ejecuta.



Para más información

Para ampliar Información sobre el uso de Scapy y sus funciones:

https://scapy.readthedocs.io/en/latest/

4. PUNTOS CLAVE

- Uno de los elementos más importantes en las redes son los sockets y en Python se encuentra disponible el módulo “socket” que precisamente permite creación y gestión de este tipo de componentes.
- Existen dos tipos de sockets: Cliente y Servidor. Ambos tipos comparten las mismas primitivas bases, en embargo el objetivo de ambos es distinto. El cliente utiliza la primitiva connect() para establecer una conexión con un servidor y el servidor las primitivas bind(), listen() y accept() para recibir conexiones.
- Scapy es una librería muy extendida a la hora de crear scripts en Python que puedan manipular paquetes de datos, capturar y reinyectar tráfico.
- Scapy puede utilizarse desde un intérprete que permite explorar sus funciones y familiarizarse con el entorno.
- Scapy permite la creación, captura y reinyección de paquetes, facilitando las labores de detección y explotación de vulnerabilidades en entornos de red.